

LUNARIS
Digital Solutions

Wenn Coding Agents schneller sind als jedes Review

Lars Roith · VDMA Expertenkreis · 11.11.2026

Code entsteht heute anders



Mensch + Copilot

Mensch schreibt, Copilot vervollständigt.

Chat / Agent

Ganze Klassen oder Features auf Zuruf.

Coding Agent

Bearbeitet ein PBI weitgehend selbstständig.

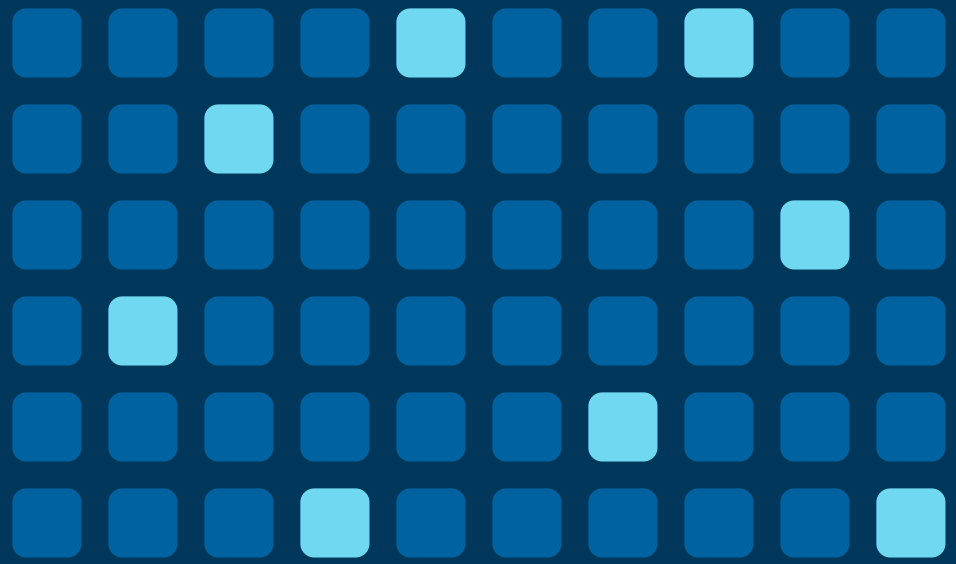
Mehrere Agents parallel

Gleichzeitige Änderungen – kein Mensch liest alle mit.

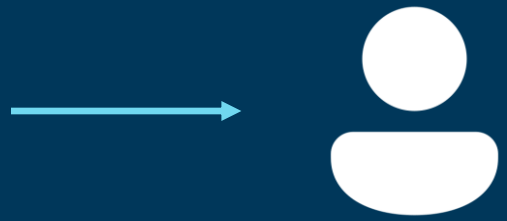
Die Codeerzeugung wird schneller – die menschliche Prüfung nicht.

LUNARIS
Digital Solutions

Code zu erzeugen ist nicht mehr der Engpass



Pull Requests



1 Reviewer

Täglich im Monorepo

30.000+

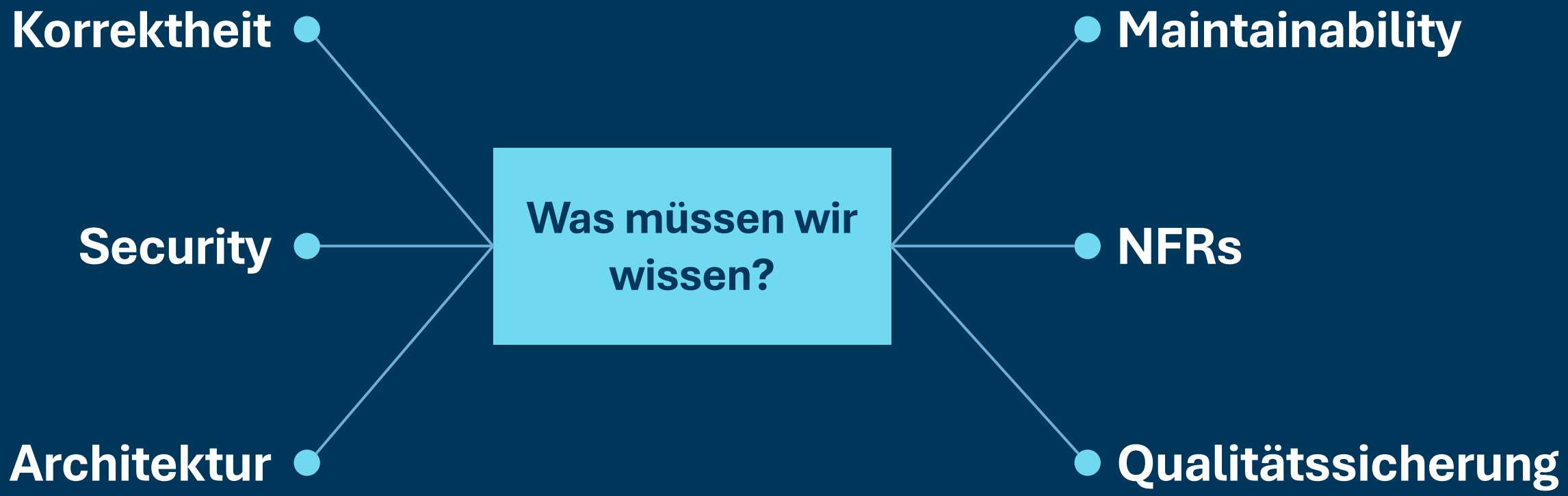
Änderungen pro Tag – zuvor ca. 2.500

Wenn niemand mehr jede Zeile liest: Was wollen wir mit einem Review eigentlich feststellen?

Quellen: Anthropic, „How the Claude Code team does code review“; Microsoft (@msdev)

Review bedeutet mehr als Code lesen

Nicht jede dieser Fragen sollte ein Mensch beantworten.



Quelle: Lunaris Digital Solutions, Artikel zum Copilot Code Review Agent (03.02.2026)

LUNARIS Digital Solutions

Nicht jede Prüfung gehört in ein Code Review

1

Maschine prüft Regeln.

BEISPIEL

Linter, Tests, Security-Scans

2

KI prüft Kontext.

BEISPIEL

Review gegen Anforderungen, DoD und Guidelines

3

Mensch verantwortet Entscheidungen.

BEISPIEL

Intention, Trade-offs, Risiko

Quellen: Microsoft (@msdev); Anthropic, „How the Claude Code team does code review“

Ebene 1: Deterministische und automatisierte Prüfung

Was sich als Regel formulieren lässt, muss niemand lesen.



Code & Stil

Prettier · ESLint



Security & Lieferkette

CodeQL · Dependabot



Verhalten & Tests

xUnit · Testcontainers



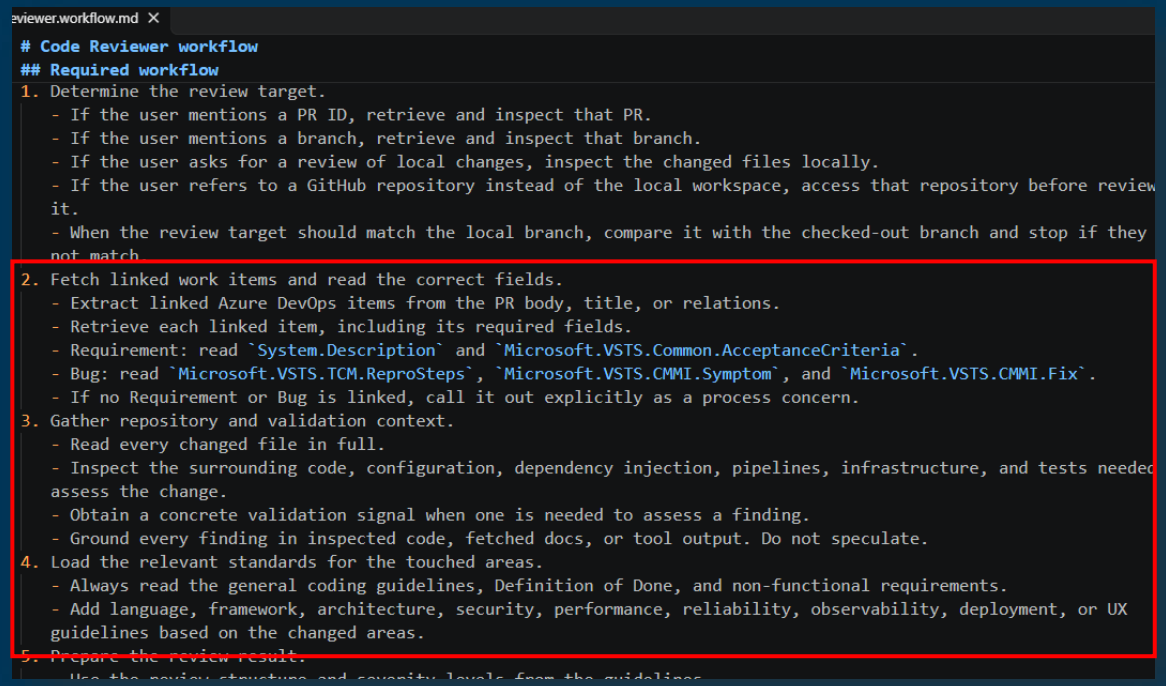
Architektur, Betrieb & Beweis

ArchUnitNET · BenchmarkDotNet

Ebene 2: KI prüft, was Kontext und Interpretation verlangt

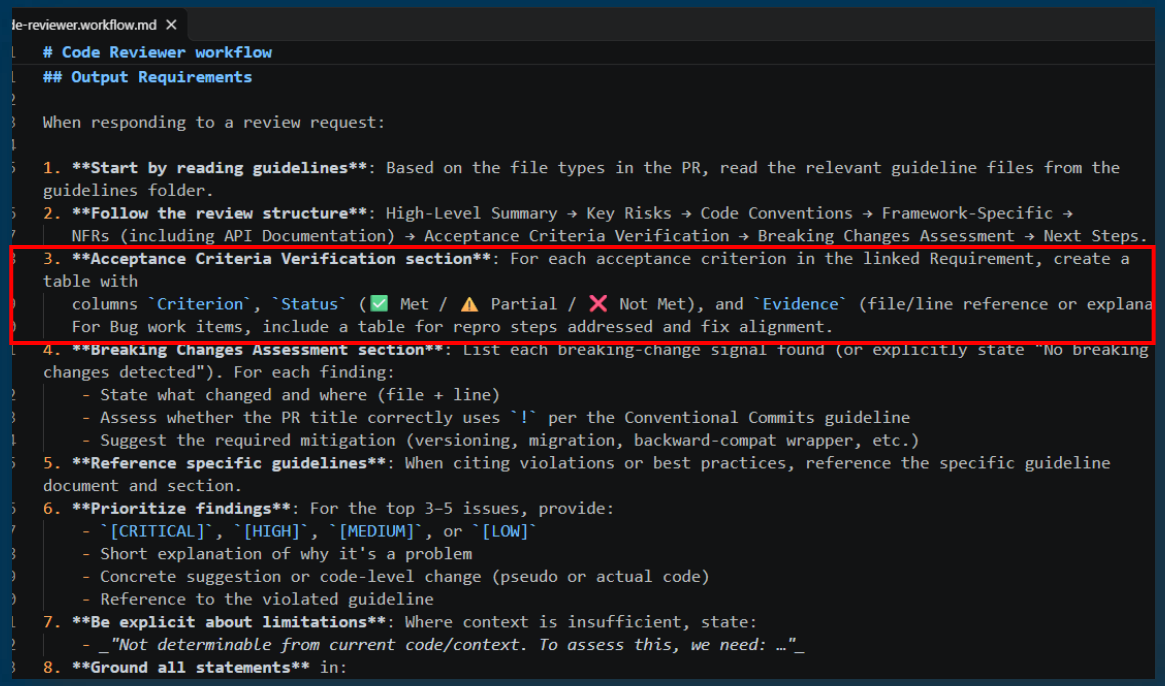
Der Agent bekommt eine verbindliche Arbeitsanweisung.

Anforderungen statt bloßer Diff-Prüfung



```
eviewer.workflow.md
# Code Reviewer workflow
## Required workflow
1. Determine the review target.
   - If the user mentions a PR ID, retrieve and inspect that PR.
   - If the user mentions a branch, retrieve and inspect that branch.
   - If the user asks for a review of local changes, inspect the changed files locally.
   - If the user refers to a GitHub repository instead of the local workspace, access that repository before review
   it.
   - When the review target should match the local branch, compare it with the checked-out branch and stop if they
   not match.
2. Fetch linked work items and read the correct fields.
   - Extract linked Azure DevOps items from the PR body, title, or relations.
   - Retrieve each linked item, including its required fields.
   - Requirement: read `System.Description` and `Microsoft.VSTS.Common.AcceptanceCriteria`.
   - Bug: read `Microsoft.VSTS.TCM.ReproSteps`, `Microsoft.VSTS.CMMI.Symptom`, and `Microsoft.VSTS.CMMI.Fix`.
   - If no Requirement or Bug is linked, call it out explicitly as a process concern.
3. Gather repository and validation context.
   - Read every changed file in full.
   - Inspect the surrounding code, configuration, dependency injection, pipelines, infrastructure, and tests needed
   assess the change.
   - Obtain a concrete validation signal when one is needed to assess a finding.
   - Ground every finding in inspected code, fetched docs, or tool output. Do not speculate.
4. Load the relevant standards for the touched areas.
   - Always read the general coding guidelines, Definition of Done, and non-functional requirements.
   - Add language, framework, architecture, security, performance, reliability, observability, deployment, or UX
   guidelines based on the changed areas.
5. Prepare the review result.
   - Use the review structure and severity levels from the guidelines.
```

Nachweise statt pauschalem „Approved“



```
le-reviewer.workflow.md
# Code Reviewer workflow
## Output Requirements

When responding to a review request:

1. **Start by reading guidelines**: Based on the file types in the PR, read the relevant guideline files from the
guidelines folder.
2. **Follow the review structure**: High-Level Summary → Key Risks → Code Conventions → Framework-Specific →
   NFRs (including API Documentation) → Acceptance Criteria Verification → Breaking Changes Assessment → Next Steps.
3. **Acceptance Criteria Verification section**: For each acceptance criterion in the linked Requirement, create a
table with
   columns `Criterion`, `Status` (✅ Met / ⚠️ Partial / ❌ Not Met), and `Evidence` (file/line reference or explana
   For Bug work items, include a table for repro steps addressed and fix alignment.
4. **Breaking Changes Assessment section**: List each breaking-change signal found (or explicitly state "No breaking
changes detected"). For each finding:
   - State what changed and where (file + line)
   - Assess whether the PR title correctly uses `!` per the Conventional Commits guideline
   - Suggest the required mitigation (versioning, migration, backward-compat wrapper, etc.)
5. **Reference specific guidelines**: When citing violations or best practices, reference the specific guideline
document and section.
6. **Prioritize findings**: For the top 3–5 issues, provide:
   - `[CRITICAL]`, `[HIGH]`, `[MEDIUM]`, or `[LOW]`
   - Short explanation of why it's a problem
   - Concrete suggestion or code-level change (pseudo or actual code)
   - Reference to the violated guideline
7. **Be explicit about limitations**: Where context is insufficient, state:
   - _"Not determinable from current code/context. To assess this, we need: …"_
8. **Ground all statements** in:
```


LUNARIS
Digital Solutions

Lokal und im Workflow: dieselbe Review-Spezifikation

Zwei Einstiege, eine verbindliche Quelle.

Lokal: Review-Agent

Code Reviewer.md

```
---
description: Deep code review for .NET and Web – focus on quality, performance, security, and NFRs.
model: GPT-5.6 Terra
tools: ['search', 'runCommands', 'runTasks', 'Azure MCP/documentation', 'Azure MCP/get_bestpractices', 'Azure MCP/
search', 'github/create_pull_request_review_comment', 'github/get_branch', 'github/get_pull_request', 'github/
list_pull_request_review_comments', 'github/list_pull_request_review_threads', 'github/list_pull_requests', 'github/
reply_to_pull_request_review_comment', 'github/list_commits', 'github/resolve_pull_request_review_thread', 'github/
search_code', 'ado/mcp_ado_search_workitem', 'ado/wit_work_item', 'usages', 'problems', 'changes', 'fetch',
'githubRepo']
---

workflow: [code-reviewer.workflow.md](../agent-resources/code-reviewer.workflow.md)

Read and follow the linked workflow exactly. It is the execution specification; carry out every applicable step.
```

Automatisiert: GitHub-Workflow

aw-code-review.md

```
# Code review

Read `../agent-resources/code-reviewer.workflow.md` and follow it for pull request
`${{ github.event.pull_request.number }}`.
The workflow is the execution specification; carry out every applicable step.

Read `../agent-resources/cloud-agent-azure-ado-cli-auth.workflow.md` and complete its prerequisites before starting
the review workflow.

Read `../agent-resources/cloud-agent-graphify-download.workflow.md` and complete its prerequisite
(download the Graphify graph) right after the Azure login above, before starting the review.

Post a concise pull request comment with the prioritized review findings or confirm that no actionable findings were
found.
Use multiple comments when needed: one summary comment plus separate comments for distinct issues.
```

code-reviewer.workflow.md

eine Review-Spezifikation für beide Einstiege

LUNARIS Digital Solutions

Qualitätsregeln gehören zum Code

```
docs/internal/docs/development/general/
├── guidelines/
│   ├── code-review.md
│   ├── architecture.md
│   ├── security.md
│   ├── performance.md
│   ├── reliability.md
│   ├── maintainability.md
│   ├── observability.md
│   ├── coding-guidelines/
│   │   ├── general.md
│   │   ├── csharp.md
│   │   └── typescript.md
│   ├── framework-guidelines/
│   │   ├── angular.md
│   │   └── asp-net-core.md
│   └── testing/
│       ├── testing.md
│       ├── unit-testing.md
│       ├── integration-testing.md
│       └── architecture-testing.md
└── process/
    ├── definition-of-done.md
    └── non-functional-requirements.md
```

Versioniert.

Gemeinsam genutzt.

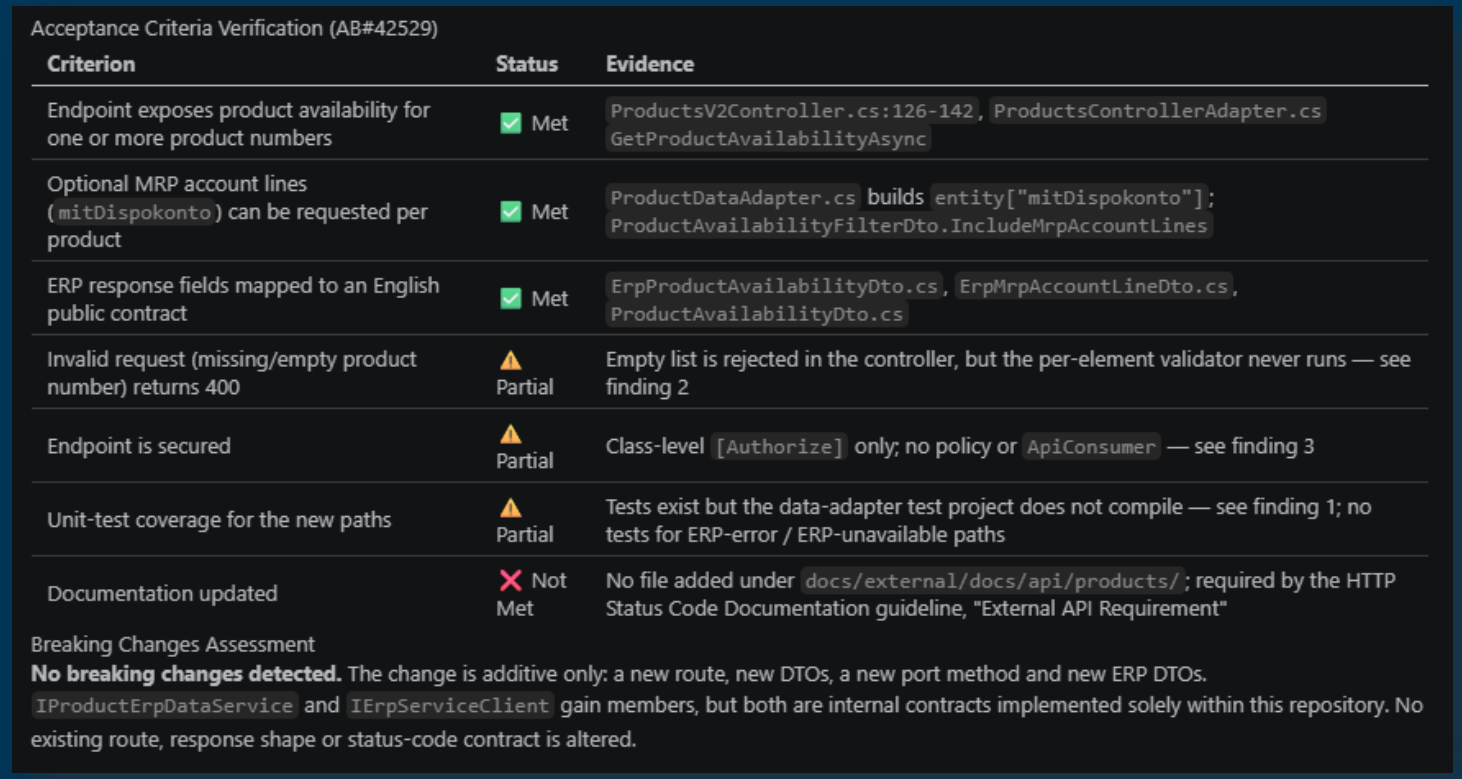
Im Review referenziert.

```
27
28  Before starting a review, collect:
29
30  - The pull request diff and commit history.
31  - The linked Requirement or Bug work item.
32  - Relevant coding guidelines for the affected languages and frameworks.
33  - Existing tests, validation results, and deployment implications.
34
35  ## Review expectations
36
37  - Base findings on evidence from the change, linked work item, and relevant guidance.
38  - Prioritize findings by impact and risk.
39  - Separate required fixes from optional improvements.
40  - Be explicit when a change is acceptable but carries follow-up risk.
41  - Call out missing tests, documentation, or operational safeguards when they matter.
42  - Re-review addressed changes until blocking findings are resolved or explicitly accepted.
43
44  ## Severity levels
45
46  Use consistent severity levels when classifying findings:
47
48  - `[CRITICAL]` for security issues, data loss risks, broken behavior, or severe performance bottlenecks.
49  - `[HIGH]` for likely bugs, scalability issues, or major maintainability risks.
50  - `[MEDIUM]` for code-quality problems, test gaps, or patterns that will create later risk.
```

Quellen: Lunaris Digital Solutions (Copilot Code Review Agent, 03.02.2026)

LUNARIS Digital Solutions

Der Review-Agent liefert Nachweise statt „Approved“



Acceptance Criteria Verification (AB#42529)

Criterion	Status	Evidence
Endpoint exposes product availability for one or more product numbers	✅ Met	`ProductsV2Controller.cs:126-142`, `ProductsControllerAdapter.cs` `GetProductAvailabilityAsync`
Optional MRP account lines (`mitDispokonto`) can be requested per product	✅ Met	`ProductDataAdapter.cs` builds `entity["mitDispokonto"]`; `ProductAvailabilityFilterDto.IncludeMrpAccountLines`
ERP response fields mapped to an English public contract	✅ Met	`ErpProductAvailabilityDto.cs`, `ErpMrpAccountLineDto.cs`, `ProductAvailabilityDto.cs`
Invalid request (missing/empty product number) returns 400	⚠️ Partial	Empty list is rejected in the controller, but the per-element validator never runs — see finding 2
Endpoint is secured	⚠️ Partial	Class-level `[Authorize]` only; no policy or `ApiConsumer` — see finding 3
Unit-test coverage for the new paths	⚠️ Partial	Tests exist but the data-adapter test project does not compile — see finding 1; no tests for ERP-error / ERP-unavailable paths
Documentation updated	❌ Not Met	No file added under `docs/external/docs/api/products/`; required by the HTTP Status Code Documentation guideline, "External API Requirement"

Breaking Changes Assessment

No breaking changes detected. The change is additive only: a new route, new DTOs, a new port method and new ERP DTOs. `IProductErpDataService` and `IErpServiceClient` gain members, but both are internal contracts implemented solely within this repository. No existing route, response shape or status-code contract is altered.

Auszug aus dem Review-Kommentar: Acceptance Criteria Verification

Kriterium.

Status.

Beleg.

- Jedes Akzeptanzkriterium einzeln bewertet: erfüllt, teilweise oder nicht erfüllt
- Belege mit Datei- und Klassenbezug statt Bauchgefühl
- Fehlende Doku wird gegen die Guideline begründet
- Offene Findings gehen an den Menschen zur Entscheidung

Quellen: Lunaris Digital Solutions (Copilot Code Review Agent, Auszug zu AB#42529)

Ebene 3: Der Mensch urteilt, wo Urteilskraft gebraucht wird

1. Welche Risiken akzeptieren wir?
2. Welche Trade-offs gehen wir ein?
3. Ist das Ergebnis für den Anwender sinnvoll?
4. Übersehen wir etwas, das nicht in der Spezifikation stand?
5. Reicht die Evidenz aus Automation und KI?
6. Können wir die Änderung verantworten?

Cognitive Surrender ist das Risiko der Rolle

Das Risiko der KI-Nutzung

Cognitive Offloading
Delegieren, selbst verantworten

Deferral
KI-Ausgabe wird zum eigenen Input

Cognitive Surrender
KI-Ausgabe wird ungeprüft übernommen

Die Regel: Entscheidungen nach Risiko und Reversibilität steuern

Zweiwegtür
Rückgängig, begrenzte Auswirkung

KI kann entscheiden
z. B. Benennung, Formatierung, Refactoring, kleine Code-Änderungen

Einwegtür
Schwer oder nicht rückgängig, hohe Auswirkung

Mensch entscheidet
z. B. Datenmigration, Änderungen an Architektur und Sicherheit, Releases, produktive Daten

Weitere Faktoren berücksichtigen

- Schadens-potenzial
- Überprüfbarkeit (Verifikation)
- Schutz-mechanismen
- Regulatorik (z. B. CRA, Datenschutz)

Konzept „Cognitive Offloading vs. Cognitive Surrender“ nach Addy Osmani (@addyosmani) auf X; Einweg-/Zweiwegtüren nach Jeff Bezos

Fazit: Wir brauchen ein explizites System of Trust

1

Vertrauen ist der Engpass, nicht Code.

Erzeugen ist billig geworden. Belegen nicht.

2

Regeln gehören ins Repo.

Versionierte Qualitätsregeln – für Team, CI und Agent.

3

Verantwortung bleibt beim Menschen.

Wir automatisieren Arbeit, nicht Verantwortung.

DREI EBENEN VON EVIDENZ



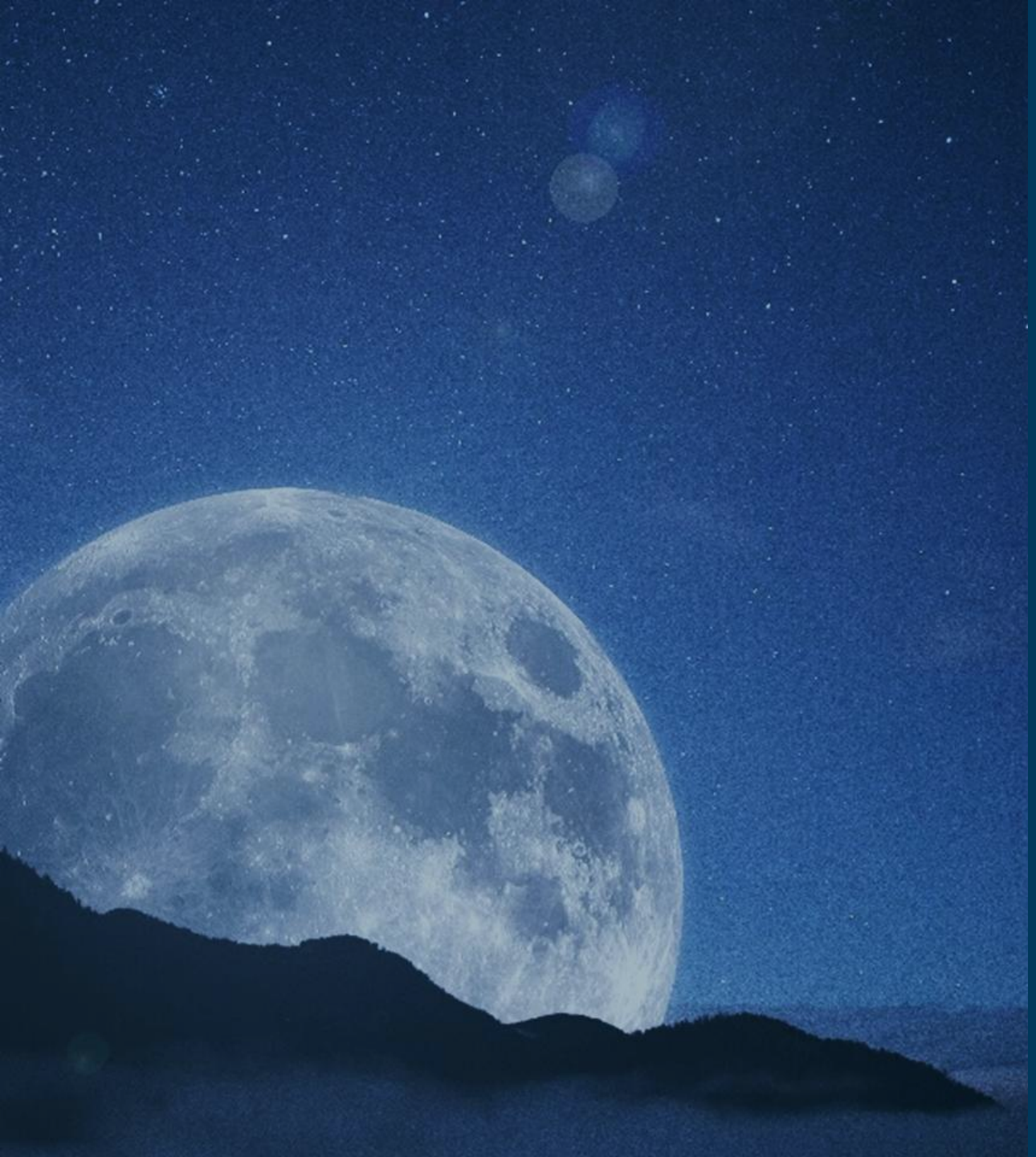
Maschine prüft Regeln.



KI prüft Kontext.



Mensch entscheidet.



FRAGEN & DISKUSSION



Lars Roith

Solution Consultant | CEO

Lars.Roith@lunaris.digital

https://lunaris.digital

LUNARIS
Digital Solutions


LUNARIS
Digital Solutions